Quality-Aware Academic Research Tool Development

Hyun Cho and Jeff Gray
Department of Computer Science
University of Alabama
Tuscaloosa, AL, U.S.A
hcho7@cs.ua.edu, gray@cs.ua.edu

Yu Sun
Department of Computer and Information Science
University of Alabama at Birmingham
Birmingham, AL, U.S.A
yusun@cis.uab.edu

Abstract— **Many organizations have adopted several different kinds of commercial software tools for the purpose of developing quality software, reducing time-to-market, and automating labor intensive and error-prone tasks. Academic researchers have also developed various types of tools, primarily as a means toward providing a prototype reference implementation that corresponds to some new research concept. In addition, academic researchers also use the tool building task itself as a mechanism for students to learn and practice various software engineering principles (e.g., requirements management, design, implementation, testing, configuration management, and release management) from building the tools. Although some academic tools have been developed with observance of sound software engineering practices, most academic research tool development still remains an ad hoc process because tools tend to be developed quickly and without much consideration for quality. In this paper, we present several quality factors to be considered when developing software tools for academic research purposes. We also present a survey of tools that have been presented at major conferences to examine the status quo of academic research tool development in terms of these factors.**

Keyword; Quality Factor, Academic Software Development, Evaluation

I. Introduction

Academic research tool development is one way to showcase and promote the experimentation of newly introduced research concepts. Many tools have been developed in academia, and some of them have been commercialized successfully. For example, Simulation Program with Integrated Circuit Emphasis (SPICE) [12] is a well-known circuit simulation tool and widely used in academia and industry. SPICE was developed at the Electronics Research Laboratory of the University of California, Berkeley, and the first paper was published in 1973. The first commercial version of SPICE was introduced in 1990 under the name of ISPICE and then Microsim announced PSPICE, which was developed based on SPICE 3. Statistical Package for the Social Sciences (SPSS) [23] is another example of successful commercialization from the foundation of an academic research tool. However, most tools disappear from focus or are no longer maintained after the research on a specific project ends. The reason for this lack of continuity of tool maturation is that academic tool development has different purposes when compared with industrial software development. The goal of academic tool development is often to implement a proof-of-concept prototype or provide a context for experimentation, rather than developing quality software for commercial use. In addition, tools tend to be developed with small budgets, inexperienced programmers (normally students), and without well-defined processes. Thus, academic tools are sometimes considered to be error-prone and difficult to install/run.

However, some researchers have proposed and experimented with improving software quality in academia by assessing the quality of academic software and teaching development process and methodology to student developers. Padua [13] introduced comprehensive quality assurance procedures for measuring the quality of course projects objectively and effectively. Other academic researchers have adopted the Team Software Process (TSP) [25], which is a comprehensive methodology for team programming proposed by the Software Engineering Institute. Some initial reports in using TSP have shown that students produce quality software successfully when following the TSP principles [4][6][9]. However, most of these efforts targeted course assignments or projects, rather than the specific case of academic research tool development, which has slightly different goals and characteristics. Both course assignments and academic research tool development require heavy student involvement. Students usually do not have sufficient knowledge about development processes or enough experiences about specific domains [1][5]. Unlike course projects, research tool development often combines one or more other research projects and has a longer lifecycle than course projects [5]. In addition, research software has higher potential to be commercialized than a short-term focused class project.

In order to emphasize the importance and highlight the key needs for developing quality research tools in academia, we propose quality factors for academic research tool development. Using these quality factors as comparison criteria, we conducted a survey of 58 tools that were published in major software engineering conferences. The purpose of this survey was to assess the status quo of academic research tool development. The rest of the paper is organized as follows: Section II introduces suggested quality factors for academic tool development from engineering and business perspectives. Section III presents the results of academic research tool assessment conducted for 58 tools, and Section IV offers a discussion of the results and concludes the paper.

II. Quality Factors For Academic Software Development

Many quality factors (e.g., performance, modifiability, and portability) affect the design and implementation of a software system and have been used to measure the success of software development. However, the selection of quality factors are subject to the context of the stakeholders so that software systems can have different sets of quality factors according to their purpose and need. In this section, we discuss quality factors, from both development and business perspectives, which we believe should be considered when developing research tools in academia.

A. Quality Factors for Software Development

From the software development perspective, relating to issues surrounding sound software engineering principles, we suggest the following quality factors:

Interoperability or Integration with other tools. Interoperability/Integration between tools is an enabler for leveraging data reuse and/or shortening development time. Thus, seamless tool interoperability/integration can assist software engineers in developing quality software quickly and cheaply. For example, when requirements of a software system are analyzed and the system needs to be designed with object-oriented analysis and design principles, a software engineer can design the system with a UML tool. To capture and verify that every requirement is modeled in the UML tool, the designer could use a UML tool (e.g., IBM Rational TAU) that can be integrated with requirements management tools (e.g., IBM Rational DOORS [21]). The integration between a requirements tool and a modeling tool can assist in understanding how the requirements are reflected and realized in the design. According to Howie et al. [7], interoperability can be classified at several levels, such as physical, data-type, specification-level, and semantic. Specification and semantic level interoperability are related to the encapsulation of knowledge representation and the design intent, respectively. With development of commercial-level software tools, much time and effort is allocated to design and implement a system to support such levels, which may be beyond the scope of academic tool development (in terms of schedule and available resources). Although all tools cannot support such levels of interoperability, tools should support at least data-type interoperability, which allows sharing information through electronic formats. For example, IBM Rational Rose [19] and IBM Rational TAU [20] use their own file format to maintain UML models such that they cannot load or modify files from each other. However, both tools provide a feature, called XML Metadata Interchange (XMI) [28], which is standardized by OMG for exchanging metadata information, in order to support data-type level interoperability.

Development Capability. Measuring the capability of software development is one way to estimate software quality and productivity. Several software development measurement models are available for assessing the maturity level of software development, acquisition and utilization (e.g., CMMI [24], ISO 9000-3 [26], and ISO/IEC 15504 (i.e., SPICE) [27]). In general, while spending several years in measuring the maturity level of their organization with these measurement models, organizations become more capable to develop quality software. But, for software development in academia, the desired and appropriate maturity levels are different. Collofello and Ng [3], as well as Rout and Seagrott [16] reported the assessment results from their student team projects. They used CMM and SPICE, respectively, as the assessment model. They reported that most projects achieved Level 2 capability, which indicates that the software development is stable and repeatedly manageable. Some projects reached Level 3 if they were well guided. However, they found that following formal software development processes may not be appropriate for academic software development. Many researchers [2][8] have experimented to find the appropriate development process for an academic environment. A common finding is that at least two practices should be kept during the tool development - software configuration management (SCM) and change/defect management (CM/DM). SCM plays important roles in identifying configuration items and managing baselines of the source code. Additionally, SCM supports automated build management. CM/DM manages change requests and defects. Normally, CM/DM is integrated with SCM to maintain information about how changes or defects affect the configuration items. As SCM and CM/DM cover all software development practice, they are called umbrella practices.

Source Code Quality. Many researchers have proposed various metrics to measure the quality of source code quantitatively. Uncommented Lines of Code (ULOC), McCabe Complexity [10][11], Halstead Complexity, and File/Class count are the typical metrics for source code quality measurement. Furthermore, the conformity of programming styles, which guide how to name user-defined concepts (e.g., class, methods, and variables) and how to comment the source code, is another essential factor to measure source code quality, especially regarding readability and maintainability.

B. Quality Factors from Business Perspective

In this section, we examine the quality factors that could lead to the success of academic research tools from the business perspective. Most of these suggestions are obvious candidates, but still remain uncovered by many research tool projects.

Sustainability. Many believe that the lifecycle of academic tools is short and lasts only when the research is in progress. Thus, sustainability is one of the most important factors to assure potential users that tools will be supported as long as there are users. When commercial tool vendors are being asked to present their tools, a common strategy to begin the presentation is to introduce the company including the history of the company. The goals of this introduction are advertising their company as well as stressing the sustainability of the company and its tools. Thus, to be a successful academic

Identify applicable sponsor/s here. *(sponsors)* research tool, at least a small amount of effort and time (e.g., conforming to the latest OS and language version) should be invested for sustaining the tool over years.

Technical support. Responsive and accurate technical support can improve user satisfaction greatly and help collect useful information for find defects and future enhancement of the tool. However, technical support for every request may not be feasible in academia due to the lack of time and supporters. Thus, systematic technical support such as a Frequently Asked Question (FAQ) list, and a user-supported mailing list, should be considered.

Installability. Run-time environments of end users can be different from development environments so that most commercial tools are distributed as installer packages, which contain the required run-time environments for using the tools. Considering the smaller size and complexity of research tools, distributing tools by an installer may not be necessary. However, if tools are not distributed as an installer, a sufficient amount of information should be provided in the form of an installation guide.

Licensing. Having full understanding of every license policy is unnecessary and difficult because they are written with legal terms and the interpretation may vary case by case. Normally, the research community prefers to use free and/or open source licenses for several reasons, such as sharing research results freely and shortening development time by reusing existing code. However, the use of free and open source licenses without careful consideration may cause problems if a tool is later commercialized. Thus, researchers should be careful to select the license type for their tools and understand the ramifications of licensed source code, even though they are free and open.

III. Survey And Results

A. *Survey Approach*

For assessing the maturity of academic research tool development capability and business quality, 58 tools were selected from the papers that were published over the past five years at the following conferences: the International Conference on Software Engineering (ICSE), the Foundations of Software Engineering (FSE), the Asia-Pacific Software Engineering Conference (APSEC), the European Conference on Object-Oriented Programming (ECOOP), the Object-Oriented Programming, Systems, Languages & Applications (OOPSLA), the International Conference on Objects, Models, Components and Patterns (TOOLS), and the International Workshop on Academic Software Development Tools and Techniques (WASDeTT). 83% of the tools were developed from schools in Europe and North America, and 66% of tools were introduced at ICSE, TOOLS, and WASDeTT. 26% of the tool development was supported by academic funding, and 10% was funded by both academia and industry. Table I shows the summary of tool selection.

Tool Categories. 24% of the tools were implemented for Model-Driven Engineering [18], 21% for program analysis and maintenance, 19% for new programming languages, and 13% for testing frameworks. Tools were implemented in various executable types. 53% of the tools were implemented as independent applications and 24% of the tools were developed as Eclipse plug-ins. 37% of the tools were developed in Java and 15% of the tools used C/C++ for their implementation. 72% of the tools were developed by less than 6 engineers.

To assess the maturity of software development capability and business quality, tools were examined in three categories (e.g., development capability, code quality, and business quality factor). Data was collected by retrieving information from each tool web site and reviewing contents posted on the web, as well as published papers.

Development Capability. As shown in Table I, 30 tools were developed by European academia and 18 tools were developed by academia in North America. Seven tools were presented by Asian academic researchers. Because each academic organization is geographically distributed, on-site capability assessment is not feasible. Thus, we assessed the capability by reviewing documents posted on the web including published papers. We reviewed six major documents where available: requirements, design, installation, user guide, developers guide, and project plan. The main goal of document review is to check the existence of documents instead of reviewing the contents of the documents or ranking the capability. However, because organizations provide documentation with different formats and some organizations use one or multiple files for documentation (e.g., using one file for both an installation guide and user's guide), we examined every document to check the existence of specific content. In addition, we examined the application of CM/DM and SCM as a means for assessing process management. The use of CM/DM and SCM helps to keep the integrity of the development deliverables and helps to assess the tool quality such as defect types/density and error-prone modules.

Code Quality. To measure the quality of source code, five metrics were selected: Lines of Code (LOC), which counts all lines of code including blanks and comments, McCabe's Cyclomatic Complexity, Comment to Code Ratio, Executable Statement to Code Ratio, and maximum nesting. LOC is used for estimating the required efforts of maintenance and normalizing other metrics [14]. McCabe's Cyclomatic Complexity [10][11] measures the complexity by counting the number of independent paths in a module. If a module has higher cyclomatic complexity values, it has the potential to produce more errors and be less understandable. Comment to Code Ratio is another metric to measure documentation, especially at the source code level, and comments help to improve the understandability of code and generate documents such as API references. Executable Statement to Code Ratio measures the number of actual executable lines in code and helps to understand the simplicity of the code. Maximum nesting measures the nesting level of control structures (e.g., if, while, for, switch, etc.) in a module, such that a module is regarded as hard to understand and maintain if it has a high value [17]. Because some of the selected tools have been distributed without source code, the measurement was performed on 25 out of 58 tools with Scitools Inc.'s Understand [22].

TABLE I. SUMMARY OF TOOL SELECTION BY CONFERENCE AND REGION

	APSEC	ECOOP	FSE	ICSE	OOPSLA	TOOLS	WASDeTT	Total
Asia	3			3		1		7
North America Asia		1						1
Europe	1	3	3	3	2	6	12	**30**
North America Europe							1	1
North America		5	1	5	1	2	4	**18**
South America						1		1
Total	4	9	4	**11**	3	**10**	**17**	**58**

Business Quality Factor. Because some of the business quality factors are related to the development capability, assessments for business quality factors overlap with Development Capability assessment. For example, we assume that technical support is provided through CM and email so that we examined the existence of CM as well as the assigned contacts for technical support. For assessing installability, we checked the existence of an installation guide and the type of the installation package. We consider a tool as installable if it is distributed in the form of an executable installer (e.g., Windows install package) or installed through download (e.g., Eclipse Installer). In addition, most Java-based applications are not distributed with a formal installer because most Java applications can be installed and run just by creating a directory and moving files from a zip/JAR into the directory. Finally, we checked the maintenance period of the tools for assessing sustainability and longevity of each tool.

B. Survey Result

This section provides an aggregate summary of the results of our evaluation. The specific details and evaluation of the 58 tools is available on our web site at: http://cs.ua.edu/graduate/hcho7/toolEval/data.xls. We strongly encourage the developers of the examined tools to contact us if we misunderstood the status of their tool, and we will correct this data file from any informed contact.

Development Capability. Table II shows the results of development capability assessment. As documents are the basic input for the assessment, we examined the six types of documents that are required to maintain the tools. See Table II(a). Overall, documentation seems to be neglected in academic tool development, which may not be a surprising result. In fact, over 90% of the tools examined did not have documents for requirements, design, implementation (or developers guide), or a project plan. As design and implementation documents explain the design concepts and the development environments, maintainers may need to spend a great amount of time and effort to understand the design and configure the development environment if the documents are not available. Regarding CM/DM and SCM, about 62% of the tools were developed and maintained without using any form of CM/DM and SCM, and about 21% of the tools used both CM/DM and SCM. About 31% of the tools were developed using at least one management system. See Table II(c)(d). Most academic tool development teams prefer to use SVN for their SCM tool, but various tools are used for CM/DM (e.g., GitHub or Google Code). As shown in Table II(b), about 42% of the tools were maintained for two years so that we investigated the relationship between maintenance periods and the use of CM/DM and SCM. Surprisingly, the application of CM/DM

TABLE II. RESULTS OF DEVELOPMENT CAPABILITY ASSESSMENT

(a) **Documentation**

Document Type	*Available*	*N/A*
Requirement	0	58 (100%)
Design	3	55 (94%)
Installation Guide	20	38 (66%)
User's Guide	24	34 (59%)
Developer's Guide	5	53 (91&)
Project Plan	2	56 (97%)

(b) **Maintenance Period**

Years	*Total*
< 2	24
< 5	9
<10	3
>10	2
N/A	20

(c) **RM**

RM	*Total*
Applied	18
N/A	40

(d) **SCM**

Location	*Tool*	*Total*
External	SVN	7
	Git	1
Internal	SVN	6
	CVS	2
	Git	1
N/A		41

(e) **CM/DM**

Location	*Tool*	*Total*
External	code.google	6
	Codeplex.com	1
	GitHub	2
	Kenai.com	1
	JIRA.com	1
Internal	Bugzilla	3
	Mantis	1
	Trac	1
N/A		41

and SCM is not related to the length of the maintenance period. About 80% of the groups who maintained their tools for less than three years used CM/DM and SCM. In addition, only 40% of the groups who used SCM managed their software release within an SCM tool. Regarding sustainability, 41% of the tools were maintained for less than two years, and only 5 tools are maintained over 5 years. In addition, about 63% of the tools that were maintained less than two years seemed to be developed without any funding.

Code Quality. Due to source code availability, we performed static analysis on 25 tools. The results of static code analysis are summarized in Table III. Over half of the tools were implemented with less than 50,000 lines of code, with an average of 40% of the code corresponding to comments. See Table III (a)(b). Particularly, one tool showed 1.83 in comment to code ratio, which means that there are about 2 comment lines per statement. All tools seem to have a low number for the average cyclomatic complexity, but some tools showed high numbers for maximum cyclomatic complexity. This implies that the same modules need further decomposition for better understandability and maintainability. See Table III (c)(d). In addition, many tools seemed to have complex nesting structure. About 60% of the tools showed five to ten nesting depths, which indicates that five to ten control constructs are nested in a module.

Business Quality Factor. The presence of an installation guide and the readiness of an install package were reviewed for installability. See Table III(g). 20 out of 58 tools provide installation guides and eight tools were packaged as an installer. Only 10% of the tools provide both an installation guide and distributed as an installer. Regarding licensing, 62% of the tools do not specify the license type of the tool. 64% of the tools that are licensed use one of the following three licenses: Eclipse Public License, GNU General Public License, and GNU Less General Public License.

TABLE III. RESULTS OF STATIC CODE ANALYSIS

(a) **Code Size**

Code Size (SLOC)	*Total*
< 10,000	8
< 20,000	4
< 50,000	5
<100,000	4
<200,000	3
>200,000	1

(b) **Comment to Code Ratio**

Comment/Code	*Total*
<0.3	6
<0.4	8
<0.5	7
<1.0	3
>1.0	1

(c) **Avg. Cyclomatic Complexity**

ACC	*Total*
<2.0	9
<3.0	11
<4.0	4
>5.0	1

(d) **Max. Cyclomatic Complexity**

MCC	*Total*
< 10	2
< 20	4
< 30	7
< 50	2
<100	2
>100	8

(e) **Max Nesting**

Max Nesting	*Total*
< 5	3
<10	15
<20	6

(f) **Executable Statement to Code Ratio**

Comment/Code	*Total*
<0.2	5
<0.3	15
<0.4	5

(g) **Installability**

		Installation Guide		
		N/A	Y	
Installer	N/A	36	14	50
	Y	2	6	8
		38	20	58

(h) **License**

License Type	*Total*
Apache License	2
Boost license	1
BSD license	1
Eclipse Public License	4
Eiffel Forum License	1
Erlang Public License	1
GNU General Public License	5
GNU Lesser General Public License	5
MIT License	1
Mozilla Public License	2
Private License	1

IV. Conclusion

The goal of this paper is to suggest quality factors for academic software development, especially research tool development. We presented quality factors in terms of software development and business perspectives, and then conducted a survey and assessment to understand quantitatively the current state of academic research tool development. Robillard and Robillard [15] conducted a similar survey in 1999 with a small number of course and academic projects. Although about ten years have passed since they published their survey, there seems to be little progress in the approach toward academic research tool development because our more recent findings are similar to their earlier findings.

First of all, as might be expected, documentation is largely neglected in research tool development. Over 90% of the tools examined do not have documents for requirements and design. We found some requirements and design concepts from published papers. But, as requirements and design concepts are described briefly and superficially in the papers (and the total requirements spread across several papers), it was often necessary to read all related papers to understand some of the requirements and design concepts for a tool. In addition, Ahtee and Poranen [1] showed that lack of domain knowledge and programming skill is the greatest barrier toward developing quality software for academic research tools. Thus, documentation should be emphasized when developing and maintaining quality research tools. Well-documented software can help reduce the risks that are pointed out by Ahtee and Poranen.

Regarding development process management, it does not seem to be a focus for research tool development, especially for consideration of CM/DM, and SCM. Only 31% of the tools were maintained under some form of CM/DM and SCM. Recently, many open source tools are available for CM/DM and SCM, and some companies such as Google provide these management systems for free when a development group hosts its projects on their repository. Thus, the use of CM/DM and SCM systems should be encouraged for collaborating with colleagues, managing change requests and defects. Regarding code quality, the code quality seems to be adequate, in general. However, most tools showed relatively high values in maximum cyclomatic complexity and maximum nesting, which imply that modules need to be refactored for better maintainability, testability and reusability. In terms of business quality factors, tool developers need to consider the sustainability of their tools and have a plan for technical support.

As academic research tools often continue to receive interest after their development, tools need to be maintained in an executable or buildable form over years. This is another reason for adopting CM/DM and SCM systems in academic research tool development. In addition, open-free licenses are commonly used for academic research tools, but most research groups failed to understand the importance of licensing - 62% of tools are distributed without license policies. Although full understanding of licensing is not necessary for academic research tool development, the license policy should be carefully specified anticipating any future commercialization.

While surveying academic software development, we found academic research tool development faces several issues such as lack of funding, short-term focus of research projects, and lack of software engineering experiences among student developers. To resolve the first two causes, support from outside academia is required in the form of funding and/or collaboration with industry. However, the last cause can be resolved by encouraging and monitoring development groups to follow software engineering practices and being more aware of the need for quality representation of their work.

Finally, the survey research is performed by reviewing information posted on each tool's web site and information in the published papers. Because of this inexact process of obtaining information on research tools, generalization of the results of our survey should be done with caution. However, the breadth and diversity of our study may suggest that the current state of research tool development still suffers from several quality issues.

Acknowledgment

This work is supported by NSF CAREER award CCF-1052616.

References

[1] T. Ahtee and T. Poranen, "Risks in Students' Software Projects," *In Proceedings of the 22nd Conf. on Software Engineering Education and Training (CSEET '09)*, Hyderabad, India, pp. 154-157, February 2009.

[2] R.J. Back, L. Milovanov, and I. Porres, "Software Development and Experimentation in an Academic Environment: The Gaudi Experience," *In Proceedings of the 6th International Conference on Product Focused Soft. Process Improvement (PROFES 2005)*, Oulu, Finland, Springer LNCS 3547, pp. 414-428, June 2005.

[3] J.S. Collofello and C.H. Ng, "Assessing the Process Maturity Utilized in Software Engineering Team Project Courses," *In Proceedings of Frontiers in Education Conference*, San Juan, Puerto Rico, vol.1, pp.12A9/5-12A9/9, November 1999.

[4] T. Hilburn and M. Towhidnejad, "A Case for Software Engineering," *In Proceedings of the 20th Conference on Software Engineering Education & Training (CSEET '07)*, Dublin, Ireland, pp. 107-114, July 2007.

[5] V. Hoffmann, H. Lichter and A. Nyßen, "Processes and Practices for Quality Scientific Software Projects," *In Proceedings. of the Third International Workshop on Academic Soft. Development Tools and Technique (WASDeTT-3),* Antwerp, Belgium, pp. 95-108, September 2010.

[6] W.L. Honig, "Teaching Successful 'Real-World' Software Engineering to the 'Net' Generation: Process and Quality Win," *In Proceedings of the 21st Conference on Software Engineering Education and Training (CSEET '08)*, Charleston, SC, pp.25-32, April 2008.

[7] C.T. Howie, J.C. Kunz, and K.H. Law, "Software Interoperability," Tech. Report F30602-89-C-0082, Depart. of Defense Information Analysis Center, November 1996.

[8] D. Liu, S. Xu, and M. Brockmeyer, "Investigation on Academic Research Software Development," *In Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, vol.2, Wuhan, China, pp. 626-630, December 2008.

[9] B. von Konsky and M. Robey, "A Case Study: GQM and TSP in a Software Engineering Capstone Project," *In Proceedings of the*

18th Conference on Software Engineering Education & Training (CSEET ‘05), Ottawa, Canada, pp. 215-222, April 2005.

[10] T.J. McCabe, “A Complexity Measure,” *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308-320, July 1976.

[11] T.J. McCabe and C.W. Butler, “Design Complexity Measurement and Testing,” *Communications of the ACM*, vol. 32, no. 12, pp. 1415-1425, December 1989.

[12] L.W. Nagel and D.O. Pederson, “SPICE (Simulation Program with Integrated Circuit Emphasis),” Memorandum No. ERL-M382, University of California, Berkeley, April 1973. http://www.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html

[13] W. Padua, “Using Quality Audits to Assess Software Course Projects,” *In Proceedings of the 22nd Conference on Software Engineering Education and Training (CSEET ‘09)*, Hyderabad, India, pp. 162-165, February 2009.

[14] J. Rosenberg, “Some Misconceptions About Lines of Code,” *In Proceedings of the 4th International Symposium on Software Metrics (METRICS '97),* Albuquerque, NM, pp. 137-142, November 1997.

[15] P.N. Robillard and M. Robillard, “Improving Academic Software Engineering Projects: A Comparative Study of Academic and Industry Projects,” *Annals of Software Engineering*, vol. 6, April 1999, pp. 343-363.

[16] T.P. Rout and J. Seagrott, “Maintaining High Process Capability in a Student Project Course*,” In Proceedings of the 20th Conference on Software Engineering Education & Training (CSEET ‘07)*, Dublin, Ireland, pp.37-44, July 2007.

[17] A. Schroeder, “Integrated Program Measurement and Documentation Tools,” *In Proceedings of the 7th International Conference on Software Engineering (ICSE '84)*, Orlando, FL, pp. 304-313, March 1984.

[18] D. Schmidt, “Model-Driven Engineering,” *IEEE Computer*, vol. 39, no.2, pp. 25, February 2006.

[19] IBM Rational Rose, http://www-01.ibm.com/software/awdtools/developer/rose/

[20] IBM Rational TAU, http://www-01.ibm.com/software/awdtools/tau/

[21] IBM Rational DOORS, http://www-01.ibm.com/software/awdtools/doors/

[22] SciTools Inc., Understand, http://www.scitools.com/

[23] SPSS, http://www.spss.com/corpinfo/history.htm

[24] CMMI, http://www.sei.cmu.edu/cmmi/

[25] Team Software Process (TSP), http://www.sei.cmu.edu/tsp/

[26] ISO 9000-3, http://www.iso.org/iso/catalogue_detail.htm?csnumber=26364

[27] ISO 15504, http://www.iso.org/iso/catalogue_detail.htm?csnumber=38932

[28] XMI, http://www.omg.org/spec/XMI/